GNU units program

Don Peterson 6 Jan 2011, updated 24 Oct 2011
someonesdad1@gmail.com

Unit errors have cost lots of money and lives over the centuries. One well-known example is a Mars spacecraft worth hundreds of millions of dollars that was lost because of an embarrassing mistake with units.

The GNU units program can help with unit conversion problems; it's a text-based program that runs in a console. I'll demonstrate a few of its features here. I find it's one of the most-used programs on my computer.

I'll assume you can figure out how to get the program. You may have to do the usual configure/make dance to build it from scratch. If you're on a Windows computer, you can build it with the MinGW/MSYS stuff or use a cygwin environment. There's a project on SourceForge that claims to offer a Windows build, but I haven't tried it. There is a Java GUI version of the units program here, but I haven't used it.

The bread-and-butter functionality is to convert between units (the output you see here uses the `--verbose` option):

```
You have: 25.4 cm
You want: in
        25.4 cm = 10 in
        25.4 cm = (1 / 0.1) in
```

Here, we entered 25.4 cm and asked that it be converted to inches.

The second printed number is the inverse of the conversion. This is useful in some situations. For example,

```
You have: grains
You want: pounds
        grains = 0.00014285714 pounds
        grains = (1 / 7000) pounds
```

(Note we entered just the unit, which implies a leading number of unity.) This says that 0.14 millipounds is about one grain. The inverse relationship tells us that there are exactly 7000 grains in one pound. From here on, I'll leave out the inverse conversion. I read somewhere that someone once mistook a prescription written in grains (abbreviated gr) to mean a dose in grams. The medicine was a phenobarbital and the effects could be deadly because the patient could get 15 times the required dose:

```
You have: g
You want: grain
        g = 15.432358 grain
```

You can see the base SI units that make up a composite unit:

```
You have: T
You want:
        Definition: Tesla = Wb/m^2 = 1 kg / A s^2
```

This can help you with dimensional analysis tasks. The following units are also defined in the units.dat file:

```
LENGTH              SOLID_ANGLE         RESISTANCE          LINEAR_DENSITY
AREA                MONEY               CONDUCTANCE         VISCOSITY
VOLUME              FORCE               INDUCTANCE          KINEMATIC_VISCOSI
MASS                PRESSURE            FREQUENCY           TY
CURRENT             STRESS              VELOCITY
AMOUNT              CHARGE              ACCELERATION
ANGLE               CAPACITANCE         DENSITY
```

These are also intended to help with dimensional analysis.

Another technique is to answer the `You want:` prompt with a question mark. The units program will then show you a list of all the units it knows about that you can convert the given unit to. Thus, if you enter the unit T for Tesla, you'll see the equivalent units of gauss, abvolt*sec/cm^2, and Wb/m^2. Or enter A for ampere and you can see the esu and emu units used for current that you used in the E&M class you took when dinosaurs roamed the Earth.

You can add conformable units:

```
You have: 8 weeks + 14 days + 17 hours + 22 minutes
You want: seconds
        8 weeks + 14 days + 17 hours + 22 minutes = 6110520 seconds
```

Arc measure is slightly different:

```
You have: 8 degrees + 14 arcminutes + 22 arcseconds
You want: microradians
        8 degrees + 14 arcminutes + 22 arcseconds = 143805.43 microradians
```

This shows that you can use SI prefixes as needed.

Composite units and multiplication are indicated by asterisks or spaces. Exponents are indicated by numbers following the unit (an optional ^ can be between the number and the unit). Fractions can be denoted with the | symbol. Division is indicated by the / symbol, but only one is allowed in an expression. Example: suppose a room measuring $3\frac{1}{2}$ m by $88\frac{3}{4}$ ft by 1260 inches was filled with water and drained out in 30 days, 13 hours, and 24 minutes. What was the average flow rate in cubic furlongs per fortnight?

```
You have: (3+1|2) m (88+3|4) ft 1260 in / (30 days + 13 hours + 24 minutes)
You want: furlongs3/fortnight
```

The answer is 0.00017051976. In case you don't read Jane Austen novels, a fortnight is two weeks. A furlong is 40 rods; a rod is 5.5 yards; thus, a furlong is 220 yards or an eighth of a statute mile. Now you know why they had 220 yard dashes when you were in high school -- when the parents at a track meet asked how fast their son would run his 220 yard dash, the coach could say, "Well, it won't be furlong."

You can do basic arithmetic calculations:

```
You have: (87.3*1.9 + 48.5)/69.1
You want:
        Definition: 3.1023155
```

The output is limited to 8 significant figures. Use the `-o` option to change the output formatting.

The program supports a number of nonlinear units such as temperature and American Wire Gauge:

```
You have: wiregauge(24)
You want: mm
        wiregauge(24) = 0.51055923 mm
You have: 1 mm
You want: wiregauge
        1 mm = wiregauge(18.201919)
You have: tempC(40)
You want: tempF
        tempC(40) = tempF(104)
```

Note these conversions use a functional notation. Don't confuse the above temperature conversion (which uses the expression 40*(9/5) + 32) with the following conversion of a temperature difference:

```
You have: 40 degC
You want: degF
        40 degC = 72 degF
```

which is gotten simply by multiplying 40 by 9/5:

```
You have: 40 9|5
You want:
        Definition: 72
```

The program also has some built-in elementary functions:

```
You have: 336 inches tan(17 degrees + 22 arcminutes)
You want: m
        336 inches tan(17 degrees + 22 arcminutes) = 2.6690671 m
```

The other functions are `acos`, `asin`, `atan`, `cos`, `cuberoot`, `exp`, `ln`, `log`, `log2`, `sin`, and `sqrt`. When Noah told the animals to go forth and multiply, two snakes took him aside and said, "Noah, we can't multiply -- we're adders." Fortunately, there was a picnic table made out of logs nearby, so Noah put the snakes on the table and the problem was solved.

You can define your own functions in the `units.dat` file. For examples, look at the definitions of the following functions:

```
pH(x)
tempC(x) and tempF(x)
wiregauge(x)
circlearea(r)
spherevolume(r)
spherevol(r)
square(x)
```

You can define piecewise linear functions that then subsequently use linear interpolation to find the values. See the definition of the British Standard Wire Gauge function `brwiregauge` for an example.

You can extract roots as in the following example of a fifth root (the fraction operator `|` has the highest precedence to make such expressions possible):

```
You have: 2|3^1|5
You want:
        Definition: 0.92210791
```

There are a number of constants defined in the program that can aid calculations (see the man page for more details). For example, you can calculate the height of a column of water that would have a weight that would yield a pressure of 1 atmosphere at the bottom of the column:

```
You have: atm
You want: m water
        atm = 10.33227453 m water
You have: foot water
You want: psi
        foot water = 0.4335275 psi
```

The built-in constants are:

`pi`	Ratio of a circle's circumference to diameter
`c`	Speed of light
`e`	Charge on an electron
`force`	Acceleration of gravity
`mole`	Avogadro's number
`water`	Pressure per unit height of water
`Hg`	Pressure per unit height of mercury
`au`	Astronomical unit
`k`	Boltzmann's constant
`mu0`	Permeability of vacuum
`epsilon0`	Permittivity of vacuum
`G`	Gravitational constant
`mach`	Speed of sound

The units program uses a data file `units.dat` that contains all the unit definitions. You can edit this file to add units of your own choice and remove units you don't wish to support. Using this file, the program supports many archaic units (much of the work of the units program's author and other contributors was tracking down credible definitions of these archaic units).

If you edit the `units.dat` file, it's important to run the program with the `-c` option. This checks that all the definitions reduce to primitive units (those that aren't defined in terms of any other units) and prints out any problems[1].

As an example of why you might want to edit the `units.dat` file, I use the `micro` SI prefix a lot, but prefer to use `u` for it. The default `units.dat` file defines `u` to be the atomic mass unit. It was an easy task to change all the existing occurrences of u to amu, comment out the definition of u, then add the line that defined the function `screwgauge`.

```
u-          micro
```

to let me use `u` for `micro`. Another example: US-sized number screws are often labeled as e.g. "No. 4". I wanted to be able to use no(4) to denote the associated linear dimension; it was easy to copy and modify the line for

One nice feature of open source software is that you can change its behavior should you wish. For the units program, I prefer to be able to type a 'q' in when being prompted for a number with a unit and have the program exit. It was easy to add two lines to the program's `unit.c` file to add this behavior.

Finally, a nice feature is that the program respects localization issues. An example is the gallon -- if the locale is `en_GB` (Great Britain), the definition of the gallon is changed to the British gallon, `brgallon`.

1 Another reason is to detect any circular definitions.